HW05WP_ML

July 29, 2023

1 Logistic Regression: HW #5

Table of Contents

1.1 Preliminaries

Date of analysis:

[ ]:
```
from datetime import datetime as dt
now = dt.now()
print("Analysis on ", now.strftime("%Y-%m-%d"), "at", now.strftime("%H:%M %p"))
```

```
Analysis on  2023-07-29 at 23:00 PM
```

Establish current working directory:

[ ]:
```
import os
os.getcwd()
```

[ ]:
```
'/Users/chasecarlson/Documents/GSCM Course Materials/GSCM 575 Machine Learning
in Business/Python Pjojects/GSCM-575-ML/code'
```

Import standard libraries

[ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

1.2 Read and Prepare data

A classic application of supervised machine learning classification is customer churn. The ability to successfully forecast a customer of a company's services and products about to no longer be a customer allows the company to commit resources to attempt to salvage the relationship.

The following data file contains information on over 7000 customers of a telecom service, including former customers who left the service plan within the last 30 days the data was collected. Build a model to predict customer churn (customer exits the service plan), one of the variables in the data set.

Data: http://web.pdx.edu/~gerbing/data/CustomerChurn.csv

1.2.1 Read and Verify Data

a. - Read the data into a data frame. - Display the number of rows and columns in the data file, and the first five lines of the data file, including the variable names. - Display all variable names and corresponding data values by transposing the output table. - Display the data type for each variable.

Read-in data and display first 5 rows. Number of rows and columns is displayed at the bottom.

[ ]:
```
df = pd.read_csv("http://web.pdx.edu/~gerbing/data/CustomerChurn.csv")
df.head()
```

[ ]:
```
   customerID  gender  SeniorCitizen Partner Dependents  tenure PhoneService  \
0  7590-VHVEG  Female              0     Yes         No       1           No
1  5575-GNVDE    Male              0      No         No      34          Yes
2  3668-QPYBK    Male              0      No         No       2          Yes
3  7795-CFOCW    Male              0      No         No      45           No
4  9237-HQITU  Female              0      No         No       2          Yes

      MultipleLines InternetService OnlineSecurity  … DeviceProtection  \
0  No phone service             DSL             No  …               No
1                No             DSL            Yes  …              Yes
2                No             DSL            Yes  …               No
3  No phone service             DSL            Yes  …              Yes
4                No     Fiber optic             No  …               No
```

```
  TechSupport StreamingTV StreamingMovies        Contract PaperlessBilling  \
0          No          No              No  Month-to-month              Yes
1          No          No              No        One year               No
2          No          No              No  Month-to-month              Yes
3         Yes          No              No        One year               No
4          No          No              No  Month-to-month              Yes

              PaymentMethod MonthlyCharges  TotalCharges Churn
0          Electronic check          29.85         29.85    No
1              Mailed check          56.95        1889.5    No
2              Mailed check          53.85        108.15   Yes
3 Bank transfer (automatic)          42.30       1840.75    No
4          Electronic check          70.70        151.65   Yes

[5 rows x 21 columns]
```

Display the variable names and values using transpose()

[ ]:
```
df.head().transpose()
```

[ ]:
```
                                 0             1               2  \
customerID              7590-VHVEG    5575-GNVDE      3668-QPYBK
gender                      Female          Male            Male
SeniorCitizen                    0             0               0
Partner                        Yes            No              No
Dependents                      No            No              No
tenure                           1            34               2
PhoneService                    No           Yes             Yes
MultipleLines     No phone service            No              No
InternetService                DSL           DSL             DSL
OnlineSecurity                  No           Yes             Yes
OnlineBackup                   Yes            No             Yes
DeviceProtection                No           Yes              No
TechSupport                     No            No              No
StreamingTV                     No            No              No
StreamingMovies                 No            No              No
Contract            Month-to-month      One year  Month-to-month
PaperlessBilling               Yes            No             Yes
PaymentMethod     Electronic check  Mailed check    Mailed check
MonthlyCharges               29.85         56.95           53.85
TotalCharges                 29.85        1889.5          108.15
Churn                           No            No             Yes

                                         3                 4
customerID                      7795-CFOCW        9237-HQITU
gender                                Male            Female
SeniorCitizen                            0                 0
```

```
Partner                                  No               No
Dependents                               No               No
tenure                                   45                2
PhoneService                             No              Yes
MultipleLines              No phone service               No
InternetService                         DSL      Fiber optic
OnlineSecurity                          Yes               No
OnlineBackup                             No               No
DeviceProtection                        Yes               No
TechSupport                             Yes               No
StreamingTV                              No               No
StreamingMovies                          No               No
Contract                           One year   Month-to-month
PaperlessBilling                         No              Yes
PaymentMethod     Bank transfer (automatic) Electronic check
MonthlyCharges                         42.3             70.7
TotalCharges                        1840.75           151.65
Churn                                    No              Yes
```

Display the data types for each variable:

[ ]:
```
df.dtypes
```

[ ]:
```
customerID           object
gender               object
SeniorCitizen         int64
Partner              object
Dependents           object
tenure                int64
PhoneService         object
MultipleLines        object
InternetService      object
OnlineSecurity       object
OnlineBackup         object
DeviceProtection     object
TechSupport          object
StreamingTV          object
StreamingMovies      object
Contract             object
PaperlessBilling     object
PaymentMethod        object
MonthlyCharges      float64
TotalCharges         object
Churn                object
dtype: object
```

b. The variable TotalCharges is conceptually a numeric variable but is read into the data frame as an object variable, i.e., non-numeric. Convert to numeric. As always, audit (verify) any change to *the data table.*

Use the following code for the `to_numeric` function to convert (where *d* is the data frame name, but could be any valid Python name).

`d['TotalCharges'] = pd.to_numeric(d['TotalCharges'], errors='coerce')`

The `errors` parameter set to `'coerce'` instructs to covert to a `NaN` any data value that cannot be converted to a legitimate number.

```
[ ]: df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')
```

View total missing data:

```
[ ]: df.isna().sum()
```

```
[ ]: customerID          0
     gender              0
     SeniorCitizen       0
     Partner             0
     Dependents          0
     tenure              0
     PhoneService        0
     MultipleLines       0
     InternetService     0
     OnlineSecurity      0
     OnlineBackup        0
     DeviceProtection    0
     TechSupport         0
     StreamingTV         0
     StreamingMovies     0
     Contract            0
     PaperlessBilling    0
     PaymentMethod       0
     MonthlyCharges      0
     TotalCharges       11
     Churn               0
     dtype: int64
```

1.2.2 Pre-Process Data

c. Drop the customerID variable.

Hint: Illustrated in several previous notebooks, including 02Wrangle.‘

Verify customerID was dropped by displaying the first few rows agian:

```
[ ]: df = df.drop(axis=1, columns=['customerID'])
     df.head().transpose()
```

[ ]:

```
                                 0               1                  2  \
gender                      Female            Male               Male
SeniorCitizen                    0               0                  0
Partner                        Yes              No                 No
Dependents                      No              No                 No
tenure                           1              34                  2
PhoneService                    No             Yes                Yes
MultipleLines     No phone service              No                 No
InternetService                DSL             DSL                DSL
OnlineSecurity                  No             Yes                Yes
OnlineBackup                   Yes              No                Yes
DeviceProtection                No             Yes                 No
TechSupport                     No              No                 No
StreamingTV                     No              No                 No
StreamingMovies                 No              No                 No
Contract            Month-to-month        One year     Month-to-month
PaperlessBilling               Yes              No                Yes
PaymentMethod     Electronic check    Mailed check       Mailed check
MonthlyCharges               29.85           56.95              53.85
TotalCharges                 29.85          1889.5             108.15
Churn                           No              No                Yes

                                          3                 4
gender                                 Male            Female
SeniorCitizen                             0                 0
Partner                                  No                No
Dependents                               No                No
tenure                                   45                 2
PhoneService                             No               Yes
MultipleLines              No phone service                No
InternetService                         DSL       Fiber optic
OnlineSecurity                          Yes                No
OnlineBackup                             No                No
DeviceProtection                        Yes                No
TechSupport                             Yes                No
StreamingTV                              No                No
StreamingMovies                          No                No
Contract                           One year    Month-to-month
PaperlessBilling                         No               Yes
PaymentMethod     Bank transfer (automatic)  Electronic check
MonthlyCharges                         42.3              70.7
TotalCharges                        1840.75            151.65
Churn                                    No               Yes
```

d. Most of the variables are categorical. Pre-process each categorical variable to become a dummy variable, a type of indicator variable. Retain all k dummy variables for each categorical variable with k levels (to be able to pick and choose the dummy variables to analyze.

Hint: You do not need to list each variable, though could, just the data frame name.

Use get_dummies() to transform all categorical variables in the data frame to dummy variables:

```
[ ]: df = pd.get_dummies(df)
```

e. To keep the analysis simpler, and to drop excess dummy variables retain just the following (mostly indicator) variables for analysis.

'MonthlyCharges', 'TotalCharges','PaperlessBilling_Yes', 'PaymentMethod_Mailed check', 'PhoneService_Yes', 'tenure', 'Dependents_Yes', 'InternetService_No', 'Churn_Yes'

Hint: See subsetting in 02Wrangling.

```
[ ]: subset = ['MonthlyCharges', 'PaperlessBilling_Yes', 'PaymentMethod_Mailed␣
      ↪check',
               'PhoneService_Yes', 'tenure', 'Dependents_Yes', 'InternetService_No',␣
      ↪'Churn_Yes']
     df2 = df.filter(subset)
     df2.head().transpose()
```

```
[ ]:                              0      1      2     3     4
     MonthlyCharges           29.85  56.95  53.85  42.3  70.7
     PaperlessBilling_Yes      1.00   0.00   1.00   0.0   1.0
     PaymentMethod_Mailed check 0.00  1.00   1.00   0.0   0.0
     PhoneService_Yes          0.00   1.00   1.00   0.0   1.0
     tenure                    1.00  34.00   2.00  45.0   2.0
     Dependents_Yes            0.00   0.00   0.00   0.0   0.0
     InternetService_No        0.00   0.00   0.00   0.0   0.0
     Churn_Yes                 0.00   0.00   1.00   0.0   1.0
```

In the correlation matrix later in the analysis "TotalCharges" was found to be highly correlated with MonthlyCharges. Since TotalCharges contained missing data, I dropped it from this analysis and retained MonthlyCharges instead.

f. Simplify the variable names. Rename as follows. Audit.

- MonthlyCharges –> Charges,
- PaperlessBilling_Yes –> Paperless,
- PaymentMethod_Mailed check –> Check,
- PhoneService_Yes –> Phone,
- tenure –> Tenure,
- Dependents_Yes –> Dependents,
- InternetService_No –> Internet,
- Churn_Yes –> Churn

Hint: Several previous examples, including 02Wrangle.

```
[ ]: df2.rename(columns={'MonthlyCharges': 'Charges', 'PaperlessBilling_Yes':␣
      ↪'Paperless',
```

```
                    'PaymentMethod_Mailed check': 'Check', 'PhoneService_Yes':␣
↪'Phone',
                    'tenure': 'Tenure', 'Dependents_Yes': 'Dependents',
                    'InternetService_No': 'Internet', 'Churn_Yes': 'Churn'},␣
↪inplace=True)
df2.head()
```

[ ]:

	Charges	Paperless	Check	Phone	Tenure	Dependents	Internet	Churn
0	29.85	1	0	0	1	0	0	0
1	56.95	0	1	1	34	0	0	0
2	53.85	1	1	1	2	0	0	1
3	42.30	0	0	0	45	0	0	0
4	70.70	1	0	1	2	0	0	1

To review the syntax, everything inside { } is called a Python **dictionary**, a core Python data structure. The **dictionary** lists keyword-value pairs.

g. Check for missing data. If not too much, delete the offenders. If severe, impute the missing values. Audit.

Hint: Done in 02PreProcess.

[ ]:
```
print(df2.isna().sum())
print("Total Missing: ", df2.isna().sum().sum())
```

```
Charges       0
Paperless     0
Check         0
Phone         0
Tenure        0
Dependents    0
Internet      0
Churn         0
dtype: int64
Total Missing:  0
```

No missing data after dropping the TotalCharges column, which had 11 missing.

1.2.3 Pre-Analysis Understanding and Feature Selection

1.2.4 Target Distribution

h. Check out the distribution of the target, with a frequency distribution and then the corresponding bar chart.

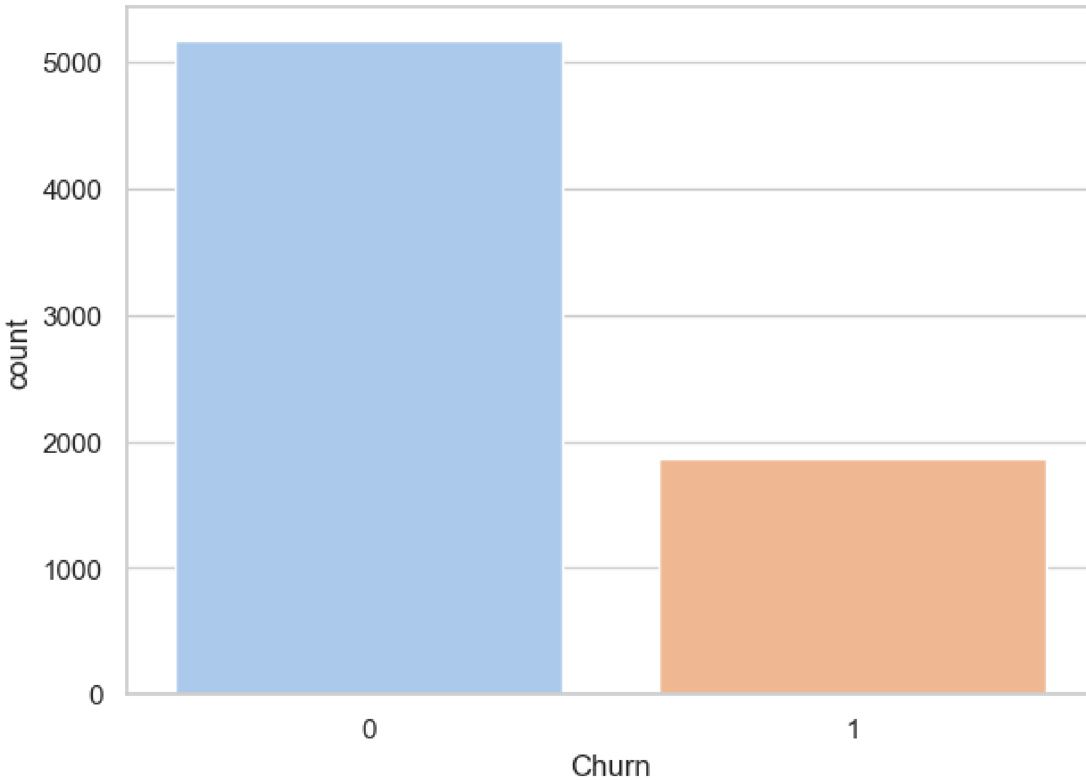
[ ]:
```
df2['Churn'].value_counts()
```

[ ]:
```
0    5174
1    1869
Name: Churn, dtype: int64
```

1,869 customers have left the service, and 5,174 have not left the service. See the bar chart below for visual representation:

[ ]:
```
sns.countplot(df2, x='Churn', palette='pastel')
```

[ ]:
```
<Axes: xlabel='Churn', ylabel='count'>
```



1.2.5 Feature Relevance

i. Are all the features relevant? Examine the difference in means of Churn across the features.

Examining the difference in means of Churn across the features, the majority of them have fairly recognizable differences except for Phone, which is 0.901 for existing customers and 0.909 for former customers.

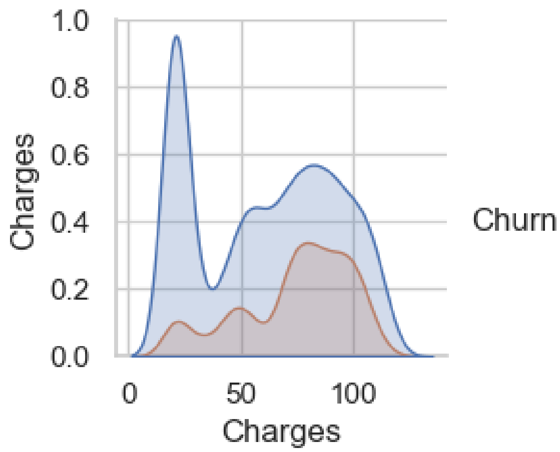
[ ]:
```
df2.groupby('Churn').mean()
```

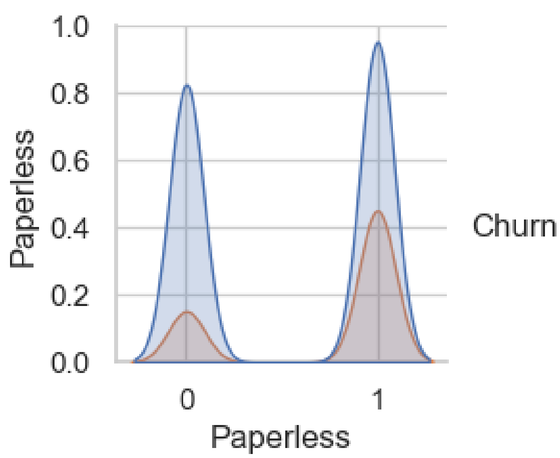
[ ]:
```
         Charges  Paperless     Check     Phone     Tenure  Dependents  \
Churn
0      61.265124   0.535562  0.252029  0.901044  37.569965    0.344801
1      74.441332   0.749064  0.164794  0.909042  17.979133    0.174425

       Internet
```

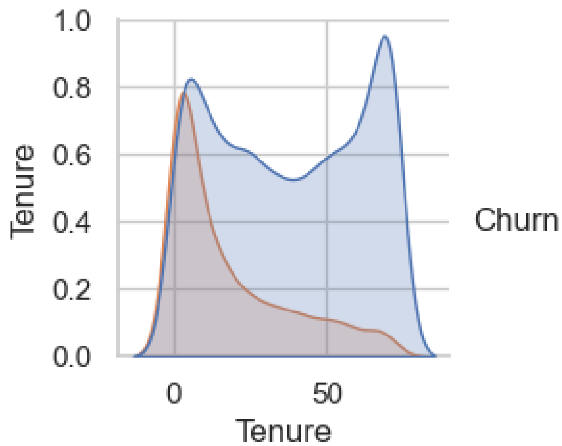
```
Churn
0       0.273096
1       0.060460
```

j. Examine the overlap in the distributions of Churn for numerical features TotalCharges, Paperless, and tenure. Which variable is likely the best predictor of churn?

[ ]:
```python
pred_vars = ['Charges', 'Paperless', 'Tenure']
for column in df2[pred_vars]:
    sns.pairplot(df2, vars=[column], hue='Churn')
```







Based on the distributions of the selected predictor variables, it appears that tenure will have the most impact on Churn rate due to having a significantly different pattern with less overlap.

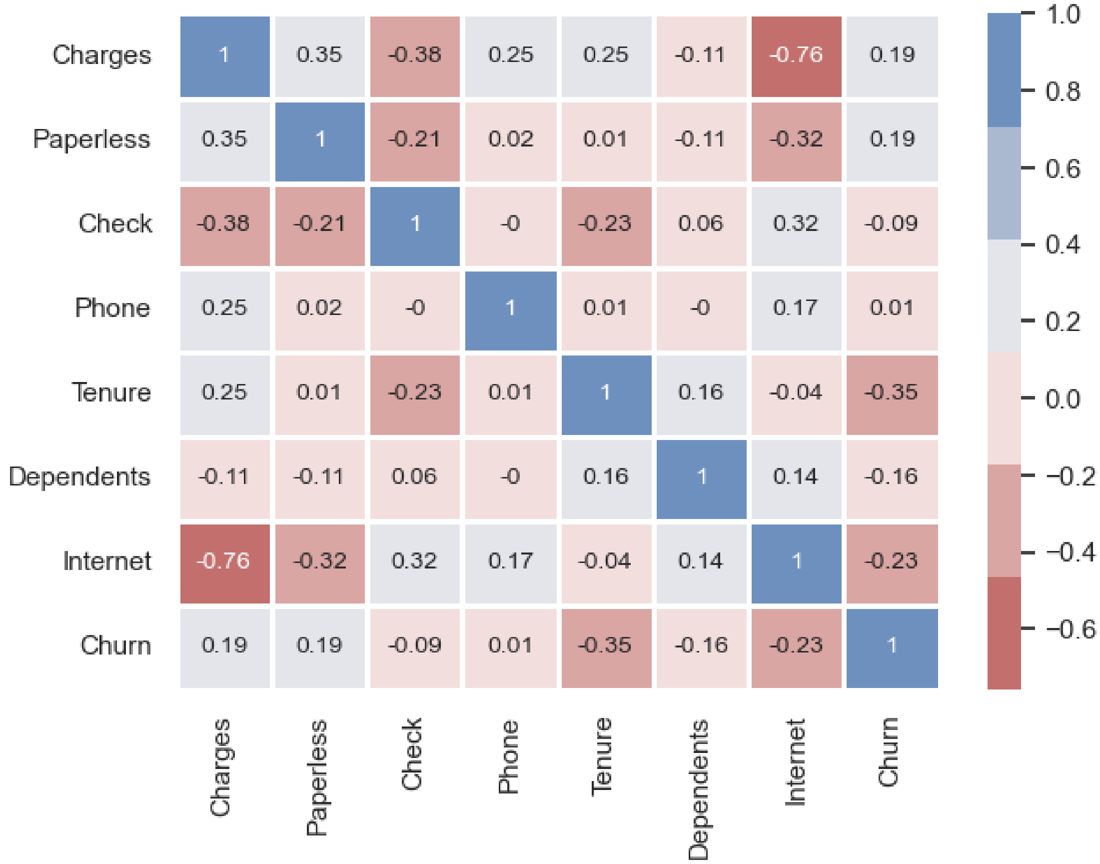
1.2.6 Feature Redundancy

k. Check for collinearity. Comment.

Because correlations span from negative to positive, use a diverging color palette, with blue indicating positive correlations and red indicating very small positive to negative correlations.

[ ]:
```
sns.set(rc={"figure.figsize": (7, 5)})
sns.heatmap(df2.corr().round(2), linewidths=2.0,
            annot=True, annot_kws={"size": 10},
            cmap=sns.color_palette("vlag_r"))
```

[ ]: <Axes: >



Based on the correlation matrix, we can identify the significant collinearity between the following variables, some of which may be dropped from the analysis: - Internet -> Charges: -0.76 - Total Charges -> Tenure: 0.83 *TotalCharges was dropped from the analysis due to changes in the HW template - Contract -> Tenure: -0.65* Contract was dropped from the analysis due to changes in the HW template - Charges -> Total_Charges: 0.65
Instead of going back to the beginning of the script to drop the columns, I could have filtered them out at this point to reduce the collinearity by manual selection.

1.2.7 Create Feature and Target Data Structures

l. Define all feature variables in a data structure X. Define the target variable as a data structure y, a column of 0's and 1's.

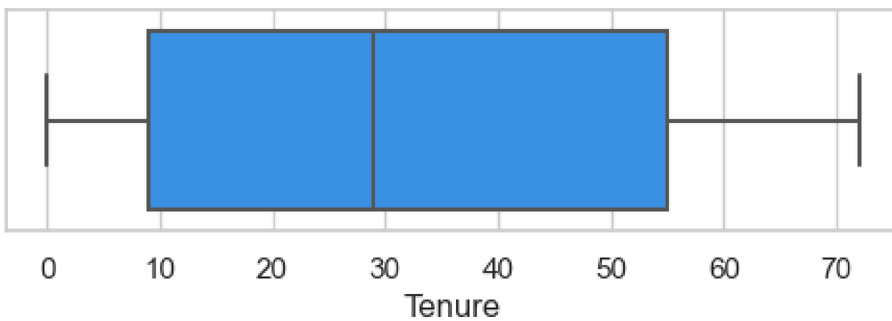
[ ]:
```
y = df2['Churn']

pred_vars = ['Charges', 'Paperless', 'Check', 'Phone', 'Tenure', 'Dependents',␣
↪'Internet']
X = df2[pred_vars]
X.shape
```

[ ]: (7043, 7)

m. All dummy variables consist of values of only 0 or 1. The numerical variables Charges *and* Tenure *range much more than 0 to 1. Generate a box plot for these variables to examine their range and check for outliers. Discuss.*

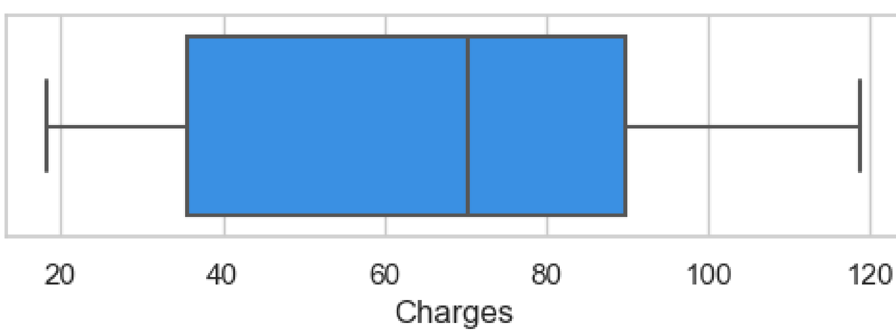
[ ]:
```
plt.figure(figsize=(6,1.5))
sns.set_theme(style='whitegrid')
sns.boxplot(df2, x='Tenure', color='dodgerblue')
```

[ ]: <Axes: xlabel='Tenure'>



[ ]:
```
plt.figure(figsize=(6,1.5))
sns.set_theme(style='whitegrid')
sns.boxplot(df2, x='Charges', color='dodgerblue')
```

[ ]: <Axes: xlabel='Charges'>



The innerquartile range of the Tenure column is about 9-55, with a median of just under 30. I do not see any outliers in either direction. The IQR of the Charges column is about 35-90, with a median value of about 70. Again, I do not see any outliers indicated outside of the whiskers in either direction.

n. Convert Charges *and* Tenure *variables to a 0 to 1 range so that all feature variables are on the same scale. As always, verify any change in the data.*

Hint: Re-scaling done in 02PreProcess.

Import the necessary package from sklearn:

[ ]:
```
from sklearn import preprocessing
```

Check data types to convert if necessary:

[ ]:
```
df2[['Charges', 'Tenure']].dtypes
```

[ ]:
```
Charges    float64
Tenure       int64
dtype: object
```

Convert Tenure to float64 and verify change:

[ ]:
```
df2.loc[:, 'Tenure'] = df2.loc[:, 'Tenure'].astype('float64')
df2[['Charges', 'Tenure']].dtypes
```

```
/var/folders/0r/8gtkp8rd5lx4q_4w2fvszkz80000gn/T/ipykernel_30884/3854154697.py:1
: DeprecationWarning: In a future version, `df.iloc[:, i] = newvals` will
attempt to set the values inplace instead of always setting a new array. To
retain the old behavior, use either `df[df.columns[i]] = newvals` or, if columns
are non-unique, `df.isetitem(i, newvals)`
  df2.loc[:, 'Tenure'] = df2.loc[:, 'Tenure'].astype('float64')
```

[ ]:
```
Charges    float64
Tenure     float64
dtype: object
```

Scale using min/max scaler

Import MinMaxScaler from sklearn and instantiate:

[ ]:
```
from sklearn.preprocessing import MinMaxScaler
mm_scaler = preprocessing.MinMaxScaler()
```

Use fit_transform() to transform Charges and Tenure columns in the df2 data frame and preview the result:

[ ]:
```
scaled_cols = ['Charges', 'Paperless', 'Check', 'Phone', 'Tenure',
               'Dependents', 'Internet']
df2[scaled_cols] = mm_scaler.fit_transform(df2[scaled_cols])
df2.head()
```

[ ]:

	Charges	Paperless	Check	Phone	Tenure	Dependents	Internet	Churn
0	0.115423	1.0	0.0	0.0	0.013889	0.0	0.0	0
1	0.385075	0.0	1.0	1.0	0.472222	0.0	0.0	0
2	0.354229	1.0	1.0	1.0	0.027778	0.0	0.0	1

```
3  0.239303         0.0     0.0     0.0   0.625000         0.0         0.0        0
4  0.521891         1.0     0.0     1.0   0.027778         0.0         0.0        1
```

Check the min & max values:

```
[ ]: print("Min Values:")
     print(df2[['Charges', 'Tenure']].min(), "\n")
     print("Max Values:")
     print(df2[['Charges', 'Tenure']].max())
```

```
Min Values:
Charges    0.0
Tenure     0.0
dtype: float64

Max Values:
Charges    1.0
Tenure     1.0
dtype: float64
```

Data Leak Warning: Better to do this analysis with the re-scaling only done on test data, then done again, anew, on the testing data separately. Otherwise, there is *data leakage*, where the testing data is confounded with the training data because at this point in the analysis, the training and testing data are together. Characteristics of the training data will impact the way that later test data is tested.

If doing just one train/test split, separate re-scaling of training and testing data can easily be accomplished with what we know. Just rescale separately the two data sets after forming the split. For the preferred k-fold cross-validation, however, the testing data in each fold needs to be re-scaled separately. To do so we need to introduce the concept of a `pipeline`, which starts to be too much after introducing everything else. To be pure, if on the job for example, should do the separate re-scaling after the train/test split and not do k-fold. Or, even better, learn about constructing a `pipeline`, such as here and here. Another straightforward step, not that hard, but enough for now and not included in this course.

Preferably, we would estimate the re-scaling parameters and then the model itself on all of the data only after successful model validation. This version of the model would then be used to forecast from new data.

Fortunately, with such a large data set, the re-scaling parameters should be reasonably robust. If not constructing a `pipeline`, this step of doing one data rescaling before validation is better than not doing any rescaling at all given the large discrepancies of scales for *TotalCharges* and *Tenure*.

1.3 Fit Model and Evaluate with *One* Hold-Out Sample

o. Do a 70% training data and 30% testing data split of X and y data structures. Show the dimensions of the output data structures.

Restablish X and y data structures due to scaling earlier:

[ ]:
```
y = df2['Churn']

pred_vars = ['Charges', 'Paperless', 'Check', 'Phone', 'Tenure', 'Dependents',␣
  ↪'Internet']
X = df2[pred_vars]
X.shape
```

[ ]:
```
(7043, 7)
```

Split the dataset and set:

[ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.30,
                                                    stratify=df2['Churn'],
                                                    random_state=1)
```

Show dimensions of output data structures:

[ ]:
```
print("Shape of X data structures: ", X_train.shape, X_test.shape)
print("Shape of y data structures: ", y_train.shape, y_test.shape)
```

```
Shape of X data structures:  (4930, 7) (2113, 7)
Shape of y data structures:  (4930,) (2113,)
```

Seven features are included in the X data structure.

p. Fit the model to the training data.

Access the solution algorithm and instantiate:

[ ]:
```
from sklearn.linear_model import LogisticRegression
logistic_model = LogisticRegression(solver='lbfgs', max_iter=500)
```

[ ]:
```
logistic_model.fit(X_train, y_train)
```

[ ]:
```
LogisticRegression(max_iter=500)
```

Evaluate fit:

[ ]:
```
y_fit = logistic_model.predict(X_train)
y_pred = logistic_model.predict(X_test)
```

q. Calculate the baseline probability for prediction in the absence of all information regarding X, the null model, the group with the largest proportion.

[ ]:
```
my = y.mean()
max_my = np.max([y.mean(), 1-y.mean()])
print('Proportion of 0\'s (no-churn): %.3f' % (1-my))
print('Proportion of 1\'s (churn): %.3f' % my)
print('Null model accuracy: %.3f' % max_my)
```

```
Proportion of 0's (no-churn): 0.735
Proportion of 1's (churn): 0.265
Null model accuracy: 0.735
```

r. As a basis for evaluating forecasting accuracy, get the values fit by the model from the corresponding X values, for training and testing data.

[ ]:
```
from sklearn.metrics import accuracy_score
print('Accuracy for training data: %.3f' % accuracy_score(y_train, y_fit))
print('Accuracy for testing data: %.3f' % accuracy_score(y_test, y_pred))
```

```
Accuracy for training data: 0.786
Accuracy for testing data: 0.794
```

s. For the testing data, calculate the probability of Churning for all the rows of testing data from the values of the features, the predictor variables.

[ ]:
```
probability = logistic_model.predict_proba(X_test)
probability = pd.DataFrame(probability)
probability.head().transpose()
```

[ ]:

	0	1	2	3	4
0	0.585594	0.45607	0.933973	0.770412	0.558314
1	0.414406	0.54393	0.066027	0.229588	0.441686

t. To understand more of what is happening here (for pedagogy), view the true values, forecasted values, and the estimated probability of Churning for about 10 or so rows of data. Best display is as a data frame.

[ ]:
```
probs = [i[0] for i in logistic_model.predict_proba(X_test)]
pred_df = pd.DataFrame({'true_values': y_test,
'pred_values': y_pred,
'pred_probs':probs})
pred_df.head(10).transpose().style.format("{:.3}")
```

[ ]:
```
<pandas.io.formats.style.Styler at 0x2931d7d30>
```

u. Assess the accuracy of the model on training and testing data. Any overfitting?

Null model accuracy: 0.735 Accuracy for testing data: 0.794 The testing performance improved 5.9% from 73.5% to 79.4%. However, at just 79.4% testing accuracy, there is significant room for improvement in forecasting accuracy. At 79.4% testing accuracy, the model is only slightly more accurate than the null model (73.5%).

v. Show the confusion matrix and explicitly identify the True Negatives, True Positives, False Negatives, and False Positives.

[ ]:
```
from sklearn.metrics import confusion_matrix
dc = pd.DataFrame(confusion_matrix(y_test, y_pred))
dc
```

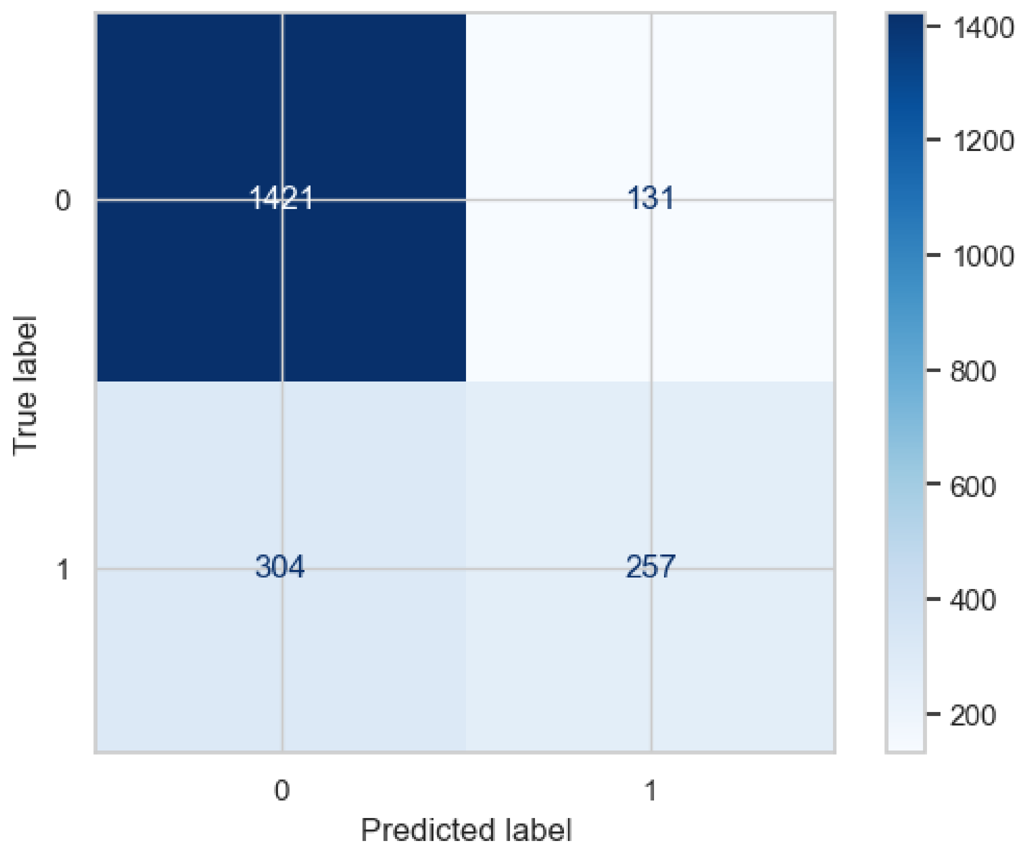
[ ]:

	0	1
0	1421	131
1	304	257

- True Negatives: 1,421
- True Positives: 257
- False Negatives: 304
- False Positives: 131

[ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay
ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap="Blues")
```

[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2930a8a90>



[ ]:
```
print("True Negatives: ", dc.iloc[0,0])
print("True Positives: ", dc.iloc[1,1])
print("False Negatives: ", dc.iloc[1,0])
print("False Positives: ", dc.iloc[0,1])
```

```
True Negatives:  1421
True Positives:  257
```

```
False Negatives:  304
False Positives:  131
```

w. Calculate the recall, precision, and, F1 metrics. Comment on the meaning of each from the perspective of management.

```
[ ]: from sklearn.metrics import recall_score, precision_score, f1_score
     print ('Recall for testing data: %.3f' % recall_score(y_test, y_pred))
     print ('Precision for testing data: %.3f' % precision_score(y_test, y_pred))
     print ('F1 for testing data: %.3f' % f1_score(y_test, y_pred))
```

```
Recall for testing data: 0.458
Precision for testing data: 0.662
F1 for testing data: 0.542
```

- Recall is a measure of the proportion of actual positive instances correctly detected as positive. A recall of .460 indicates that only 46% of the positive predictions will be correctly labeled as positive, leaving a 54% chance of misclassifying a 1 as a 0 (customer churn as a customer retained). There were 303 false negatives in the confusion matrix.
- Precision is a measure of the proportion of correctly labeled positive outcomes. A precision score of 0.662 indicates that of those that the model forecasted as churn, 66.2% are actually churn, and 33.8% are not churn (132 false positives).
- The F1 score is the balance betwen Recall and Precision, and provides the harmonic mean.

1.4 Fit Model, then Predict, Evaluate with ***Multiple*** Hold-Out Samples

x. Do a 5-fold cross-validation an report individual fold and average values of accuracy, recall, and precision.

Access the StratifiedKFold algorithm and instantiate with 5 splits:

```
[ ]: from sklearn.model_selection import StratifiedKFold
     skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)
```

Use cross_validate() to generate the scores:

```
[ ]: from sklearn.model_selection import cross_validate
     scores = cross_validate(logistic_model, X, y, cv=skf,
     scoring=('accuracy', 'recall', 'precision'),
     return_train_score=True)
```

Display the scores as a data frame:

```
[ ]: ds = pd.DataFrame(scores).round(3)
     ds.head()
```

```
[ ]:    fit_time  score_time  test_accuracy  train_accuracy  test_recall  \
     0     0.142       0.002          0.793           0.789        0.492
     1     0.016       0.002          0.783           0.791        0.414
     2     0.011       0.011          0.791           0.787        0.495
     3     0.046       0.018          0.798           0.787        0.509
```

```
4     0.021       0.015          0.777           0.793        0.457

   train_recall  test_precision  train_precision
0         0.471           0.646            0.638
1         0.476           0.640            0.642
2         0.470           0.638            0.634
3         0.467           0.651            0.635
4         0.487           0.606            0.647
```

Average values for fit metrics:

[ ]:
```
print('Mean of test accuracy: %.3f' % ds['test_accuracy'].mean())
print('Mean of test recall: %.3f' % ds['test_recall'].mean())
print('Mean of test precision: %.3f' % ds['test_precision'].mean())
```

```
Mean of test accuracy: 0.788
Mean of test recall: 0.473
Mean of test precision: 0.636
```

y. Comment on the worth of the model.

The results from the K-fold analysis are split. Accuracy dropped slightly from 0.794 to 0.788, recall improved slightly from 0.460 to 0.478, and precision dropped slightly from 0.662 to 0.635. Overall, I would suggest that this model is not highly effective at predicting customer churn due to the high probability of predicing false positives and false negatives.

1.5 Automated Feature Selection

1.5.1 Univariate Selection

z. Do a univariate feature selection of the top 4 features. Identify these features.

[ ]:
```
from sklearn.feature_selection import SelectKBest, f_classif
selector = SelectKBest(k=4).fit(X,y)
selected = selector.get_support()
selected
```

[ ]:
```
array([ True,  True, False, False,  True, False,  True])
```

Reduce the X data structure to include only the top 4 features selected into a new data structure, X2:

[ ]:
```
X2 = X.iloc[:, selected]
X2.head()
```

[ ]:

	Charges	Paperless	Tenure	Internet
0	0.115423	1.0	0.013889	0.0
1	0.385075	0.0	0.472222	0.0
2	0.354229	1.0	0.027778	0.0
3	0.239303	0.0	0.625000	0.0
4	0.521891	1.0	0.027778	0.0

The top 4 features using the univariate selection process include: Charges, Paperless, Tenure, and Internet.

1.5.2 Multivariate Selection

aa. Do a multivariate feature selection. Identify the selected features.

[ ]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import RFE
selector = RFE(logistic_model, n_features_to_select=4, step=1).fit(X,y)
```

[ ]:
```
print(selector.support_)
```

```
[ True  True False  True  True False False]
```

Reduce the X data structure to include only the top 4 features selected into a new data structure, X3:

[ ]:
```
X3 = X.iloc[:, selector.support_]
X3.head()
```

[ ]:

	Charges	Paperless	Phone	Tenure
0	0.115423	1.0	0.0	0.013889
1	0.385075	0.0	1.0	0.472222
2	0.354229	1.0	1.0	0.027778
3	0.239303	0.0	0.0	0.625000
4	0.521891	1.0	1.0	0.027778

The top 4 features using the multivariate selection process include: Charges, Paperless, Phone, & Tenure

ab. Rank the features in importance.

[ ]:
```
rnk = pd.DataFrame()
rnk['Feature'] = X.columns
rnk['Rank']= selector.ranking_
rnk.sort_values('Rank').transpose()
```

[ ]:

	0	1	3	4	5	2	6
Feature	Charges	Paperless	Phone	Tenure	Dependents	Check	Internet
Rank	1	1	1	1	2	3	4

No validation of the reduced model as the full model did not validate. Knowing the most important features serves as a building block to future models, not to serve as model on its own.

[ ]:
```
scores = cross_validate(logistic_model, X3, y, cv=skf,
scoring=('accuracy', 'recall', 'precision'),
return_train_score=True)
ds = pd.DataFrame(scores).round(3)
print(ds)
print('\n')
```

```
print('Mean of test accuracy: %.3f' % ds['test_accuracy'].mean())
21
print('Mean of test recall: %.3f' % ds['test_recall'].mean())
print('Mean of test precision: %.3f' % ds['test_precision'].mean())
```

```
   fit_time  score_time  test_accuracy  train_accuracy  test_recall  \
0     0.038       0.002          0.791           0.785        0.468
1     0.006       0.002          0.779           0.789        0.414
2     0.004       0.002          0.785           0.788        0.449
3     0.016       0.002          0.796           0.785        0.488
4     0.010       0.002          0.781           0.789        0.452

   train_recall  test_precision  train_precision
0         0.454           0.646            0.632
1         0.460           0.625            0.642
2         0.453           0.634            0.642
3         0.454           0.655            0.634
4         0.466           0.619            0.641


Mean of test accuracy: 0.786
Mean of test recall: 0.454
Mean of test precision: 0.636
```

1.5.3 Estimate Validated Model on All Data

Not going to be using this model in the future in its current form, but for completeness, provide the best estimate of the model from all of the data.

[ ]:
```
logistic_model.fit(X3, y)
```

[ ]:
```
LogisticRegression(max_iter=500)
```

1.6 Apply the Model

ac. Forecast if the customer churns from new data.

Customer data:

- Charges: 200
- Paperless: 1
- Check: 1
- Phone: 1
- Tenure: 12
- Dependents: 0
- Internet: 0

Create a list of these data values, making sure to enter in the same order that the variables appear in the X data frame.

Also, because the data was re-scaled, any new data from which to make a prediction also needs to be re-scaled. I do not believe there was an example of this, so the re-scaling transformation is provided here. Basically, take the *mm_scaler* construct previously defined from the original transformation, and then apply the `transform()` function by itself, without the `fit()` function.

[ ]:
```
X_new = [[200, 1, 1, 1, 12, 0, 0]]

X_new = mm_scaler.transform(X_new)

X_new
```

```
/Users/chasecarlson/anaconda3/envs/GSCM575-env/lib/python3.10/site-
packages/sklearn/base.py:439: UserWarning: X does not have valid feature names,
but MinMaxScaler was fitted with feature names
  warnings.warn(
```

[ ]:
```
array([[1.80845771, 1.        , 1.        , 1.        , 0.16666667,
        0.        , 0.        ]])
```

Now from this re-scaled list, create the forecast, Group 0 (not-churn) or Group 1 (churn) and the associated probability.

Reestablish the regression model with the full data set and all predictor variables:

[ ]:
```
logistic_model.fit(X, y)
```

[ ]:
```
LogisticRegression(max_iter=500)
```

Run the model with the new variables:

[ ]:
```
y_new = logistic_model.predict(X_new)
print("Predicted group membership:", y_new)
y_prob = logistic_model.predict_proba(X_new)
print(round(y_prob[0,1], 3))
```

```
Predicted group membership: [1]
0.97
```

```
/Users/chasecarlson/anaconda3/envs/GSCM575-env/lib/python3.10/site-
packages/sklearn/base.py:439: UserWarning: X does not have valid feature names,
but LogisticRegression was fitted with feature names
  warnings.warn(
/Users/chasecarlson/anaconda3/envs/GSCM575-env/lib/python3.10/site-
packages/sklearn/base.py:439: UserWarning: X does not have valid feature names,
but LogisticRegression was fitted with feature names
  warnings.warn(
```

The probability of this person being a member of group 1 is 97%, so label as Churn.